Data Storage and Formats

Relational algebra

SQL part I

Lecture 4

Fall 2008

Anna Pagh

Some figures are borrowed from the ppt slides from the book used in the course. Database Systems by Kathy Benekos, Ulises. Copyright © 2006 Prentice, Adison-Wesley, all rights reserved.

Today’s lecture

• Based on Chapter 5, pages 127-157 + 164-168.
• Relational algebra
• SQL
• Next week we continue with the rest of Chapter 5.

SQL and relational algebra

• Both SQL and relational algebra are relational query languages.
• SQL is declarative: Describe what you want
• Relational algebra is procedural: Describe how to get what you want
• Relational algebra is a higher level procedural language than e.g. Java.

Query tree

SELECT P.Name
FROM PROFESSOR P, TEACHING T
WHERE P.Id = T.ProfId
AND T.Semester = 'F1994'
AND P.DeptId = 'CS'

Query optimization

SQL is converted to relational algebra. The query optimizer decides a query execution plan.

Relational algebra

• Operators operate on relations and produce a relation as a result
• It is an algebra with 5 basic operators
  – Select
  – Project
  – Union
  – Set difference
  – Cartesian product
• Operators operates on one or two relations.
Select

- Selection of a subset of the tuples in a relation fulfilling a condition
- Denoted $\sigma_{\text{condition}}(\text{relation})$
- Operates on one relation

\[ \sigma_{\text{DeptId}=\text{CS}}(\text{PROFESSOR}) \]

Selection condition

$\sigma_{\text{condition}}(\text{relation})$

- Expression involving
  - $=$, $\neq$, $>$, $\geq$, $<$, $\leq$
  - involving two attributes or
  - one attribute and one constant
- Combination of the above using
  - AND, OR, and NOT
- Attributes must come from the relation
- Common extension is to allow string comparisons in condition

Result of selection

- The result of a selection is a relation
- The resulting relation has the same attributes as the relation that the selection operates on

\[ \begin{array}{lll}
\text{Id} & \text{Name} & \text{DeptId} \\
345 & \text{Mary} & \text{CS} \\
123 & \text{Peter} & \text{SS} \\
567 & \text{John} & \text{CS} \\
\end{array} \]

\[ \sigma_{\text{DeptId}=\text{CS}}(\text{PROFESSOR}) \]

Examples

\[ \begin{array}{lll}
\sigma_{\text{Regnr}>45}(\text{Cars}) \\
\sigma_{\text{Color} \neq \text{'Pink'}} \text{ AND Regnr}>\text{OwnerId}(\text{Cars}) \\
\sigma_{\text{Firstname}>\text{Lastname}}(\text{Owner}) \\
\sigma_{\text{Firstname} \text{ LIKE} \text{'A*'}}(\text{Owner}) \\
\sigma_{\text{Regnr}>2*\text{OwnerId}(\text{Cars})} \\
\end{array} \]

Duplicates

Removing some attributes may result in duplicate tuples, when the key is not in the attributes of the attribute list.

Since relations are sets, duplicates are removed.

\[ \begin{array}{lll}
\text{Id} & \text{Name} & \text{DeptId} \\
345 & \text{Mary} & \text{CS} \\
123 & \text{Peter} & \text{SS} \\
567 & \text{John} & \text{CS} \\
\end{array} \]

\[ \pi_{\text{DeptId}}(\text{PROFESSORS}) \]
Removing some attributes may result in duplicate tuples, when the key is not in the attributes of the attribute list. Since relations are sets, duplicates are removed.

Note! This is not true in SQL!

There may be duplicates in tables, e.g. due to projection.

Set operations - examples

\[ \pi_{\text{Color}=\text{Pink}}(\text{Cars}) \cup \pi_{\text{Color}=\text{Green}}(\text{Cars}) \]

All pink and green cars

\[ \pi_{\text{Id}}(\text{PROFESSORS}) \cap \pi_{\text{Id}}(\text{STUDENT}) \]

All Id’s of professors for which there is a student with the same id.

Set operation

Set operations are union (R ∪ S), set difference (R - S), and intersection (R ∩ S).

Set operations - examples

All pink and green cars

All Id’s of professors for which there is a student with the same id.

Problem session

Assume that we have the relations (as in KBL):

\[ \text{TRANSCRIPT} (\text{StudId}, \text{CrsCode}, \text{Semester}, \text{Grade}) \]

\[ \text{TEACHING} (\text{ProfId}, \text{CrsCode}, \text{Semester}) \]

What do the following relational algebra expressions mean?

\[ \pi_{\text{CrsCode}, \text{Semester}, \text{Grade}}(\pi_{\text{CrsCode}=\text{MAT120}}(\text{TRANSCRIPT})) \]

\[ \pi_{\text{CrsCode}, \text{Semester}, \text{Grade}}(\pi_{\text{CrsCode}=\text{MAT120}}(\text{TEACHING})) \]

\[ \pi_{\text{CrsCode}, \text{Semester}, \text{Grade}}(\pi_{\text{CrsCode}=\text{MAT120}}(\text{TRANSCRIPT})) \cap \pi_{\text{CrsCode}, \text{Semester}, \text{Grade}}(\pi_{\text{CrsCode}=\text{MAT120}}(\text{TEACHING})) \]

Cartesian product

(aka. cross product)

\[ R \times S \] for relations R and S is the relation containing all tuples that can be formed by concatenation of a tuple from R and a tuple from S.

Cartesian product

- \( R \times S \) for relations R and S, where R has n tuples and S has m tuples contain n*m tuples.
- Computationally expensive!
- Renaming necessary when R and S have attributes with the same name.
- Renaming is denoted by [name1,...] after the expression.
**Join**

\[ R \bowtie_{joincondition} S \]

The join condition has the form:

\[(R.A_1 \text{ oper}_1 S.B_1) \text{ AND } \ldots \text{ AND } (R.A_n \text{ oper}_n S.B_n)\]

where

- \(A_1, \ldots, A_n\) and \(B_1, \ldots, B_n\) are attributes for \(R\) and \(S\) respectively and
- \(\text{oper}_1, \ldots, \text{oper}_n\) are comparison operators

The join operator is equivalent to

\[ \sigma_{joincondition}(R \times S) \]

---

**Join example**

```
SELECT *
FROM Cars C, Owners O
WHERE C.Ownerid=O.Id

Cars \bowtie_{Ownerid=Id} Owners
\sigma_{Ownerid=Id}(Cars \times Owners)
```

---

**Equi-join**

- Common special case of join where all comparisons are equalities is called **equi-join**.
- The way to find out how to connect information distributed in different relations, about e.g. a person

\[ \pi_{\text{CrsName}, \text{Name}}(\{\text{PROFESSOR} \bowtie_{\text{Id}=\text{ProfId}}
\sigma_{\text{Semester}=\text{F1995}}(\text{TEACHING})
\wedge \text{CrsCode}=\text{CrsCode} \cdot \text{COURSE}) \]

---

**Natural join**

- A join where all attributes with the **same name** in the two relations are included in the join condition as **equalities** is called **natural join**.
- The resulting relation only includes one copy of each attribute.
- Natural join is denoted:

\[ R \bowtie S \]

---

**Division**

**Division** is the relation containing tuples in \(R\) that match all tuples in \(S\).

\[ R/S \]

- \(R\) has attributes \(A_1, \ldots, A_n\) and \(B_1, \ldots, B_m\)
- \(S\) has attributes \(B_1, \ldots, B_m\)
- \(R/S\) has attributes \(A_1, \ldots, A_n\)
- \(<a> \in R/S\) if and only if \((<a>) \in S \subseteq R

---

**Problem session**

What does the following expression compute?

\[ \text{ProfCourses}/\text{ProfCS} \]

where

\[ \text{ProfCS} = \pi_{\text{Id}}(\sigma_{\text{DeptId}=\text{CS}}(\text{PROFESSOR}) \wedge \text{ProfCourses}) \]

\[ \pi_{\text{ProfId}, \text{CrsCode}}(\text{TEACHING})[\text{Id, CrsCode}] \]
Query optimization

Why is it better to transform an expression to another equivalent expression?

Examples of heuristic rules:
• Join is more efficient than cross product
• Better to do selection before cross product than vice versa, when the selection reduces the relation significantly
• Better to join small relations before large relations

SQL: SELECT-FROM-WHERE

SELECT C.Color
FROM Car C
WHERE C.Regnr=42

A join query

Join

SELECT C.Color, O.Id
FROM Car C, Owner O
WHERE C.OwnerId=O.Id AND C.Regnr=42

πColor,Id(σRegnr=42(Car) ×Ownerid=1d Owner)
πColor,Id(σRegnr=42 AND Ownerid=1d(Car × Owner))

Evaluation algorithm

Algorithm for evaluating a SELECT-FROM-WHERE statement:
1. FROM clause is evaluated. Cartesian product of relations is computed.
2. WHERE clause is evaluated. Rows not fulfilling condition are deleted.
3. SELECT clause is evaluated. All columns not mentioned are removed.

A way to think about evaluation, but in practice more efficient evaluation algorithms are used.

Problem session

Assume that we want to compute the join of two relations with n tuples each and that the result contains m tuples. The join condition is an equality.

How can the join be computed without computing the cartesian product in the following two cases:
1. When there is an index on the join attribute
2. If we start by sorting the two relations wrt. the join attributes

Duplicates in tables

SELECT C.OwnerId
FROM Car C
WHERE Color='Green'

Returns multiset that has an element of multiplicity > 1 if someone owns more than one green car.
Multiset: May contain more than one tuple with the same values.

SELECT DISTINCT C.OwnerId removes duplicates.
Note that duplicate removal may be an expensive operation.
Expressions in WHERE clause

- Comparisons
  - =, <>, <=, <, >, >=, 
  - strings and numbers
  - +, -, *, / etc.
  - string operations (next slide)
- attributes from relations mentioned in FROM clause
- AND, OR, NOT
- IN (Firstname IN ('Line', 'Stine', 'Tine'))

Expressions in SELECT

You can define new attributes using expressions:

```sql
SELECT C.Ownerid, T.Amount/12
FROM Car C, Cartax T
WHERE C.Color='Green' AND C.Regnr=T.Regnr
```

You can give attributes new names:

```sql
SELECT C.Ownerid AS Id,
   T.Amount/12 AS MonthlyTax
```

Set operations

- UNION (∪), INTERSECT (∩), and EXCEPT (−)

```sql
(SELECT * FROM Car C WHERE C.Color='green')
UNION
(SELECT * FROM Car C WHERE C.Color='blue')
EXCEPT
(SELECT *
FROM Car C
WHERE NOT C.Regnr=1234)
```

Aggregate functions

- COUNT(DISTINCT attribute)
  - Counts the number of values in column attribute of the query result. The optional keyword DISTINCT indicates that each value should be counted only once, even if it occurs multiple times in different rows.
- SUM(DISTINCT attribute)
  - Sums up the values in column attribute. DISTINCT means that each value should contribute to the sum only once, regardless of how often it occurs in column attribute.
- AVG(DISTINCT attribute)
  - Computes the average of the values in column attribute. DISTINCT means that each value should be used only once.
- MAX(attribute)
  - Computes the maximum value in column attribute. DISTINCT is not used with this function, as it would have no effect.
- MIN(attribute)
  - Computes the minimum value in column attribute. Again, DISTINCT is not used with this function.
When more than one value should be computed, e.g. the total amount of tax each owner has to pay, use grouping together with aggregation:

```
SELECT C.OwnerId AS Id, SUM(T.Amount) AS TotalTax
FROM Cartax T, Car C
WHERE T.Regnr=C.Regnr
GROUP BY C.OwnerId
```

The resulting columns can only be the aggregate or columns mentioned in the GROUP BY clause.

```
SELECT C.OwnerId AS Id, SUM(T.Amount) AS TotalTax
FROM Cartax T, Car C
WHERE T.Regnr=C.Regnr
GROUP BY C.OwnerId
```

```
<table>
<thead>
<tr>
<th>OwnerId</th>
<th>Regnr</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1</td>
<td>300</td>
</tr>
<tr>
<td>1234</td>
<td>2</td>
<td>450</td>
</tr>
<tr>
<td>4321</td>
<td>3</td>
<td>210</td>
</tr>
<tr>
<td>8888</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>8888</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>1234</td>
<td>750</td>
<td></td>
</tr>
<tr>
<td>4321</td>
<td>210</td>
<td></td>
</tr>
<tr>
<td>8888</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>
```

Next lecture:

**Nested queries**

```
SELECT C.Regnr
FROM Car C
WHERE C.Color IN
    (SELECT C2.Color
     FROM Car C2
     WHERE C2.Regnr=1234 OR C2.Regnr=4321)
```

Comments and documentation

```
-- Comment describing statement
SELECT DISTINCT C.OwnerId
FROM Car C
-- Another comment
WHERE Color='Green'
```

Hand-ins should be commented, both in the code and in other documentation, e.g. description of choices made, if something is lacking etc.

**Naming conventions**

- **KBL:**
  - Uppercase for reserved words and tables
  - Uppercase for first letter in attributes
  - Tables singular, attributes singular
- **Other example:**
  - Uppercase reserved words
  - Lower case all other words
  - Tables plural, since it is a set
  - Attributes singular, since it is one value
- **My recommendation to you:**
  - Choose your own convention and be consistent